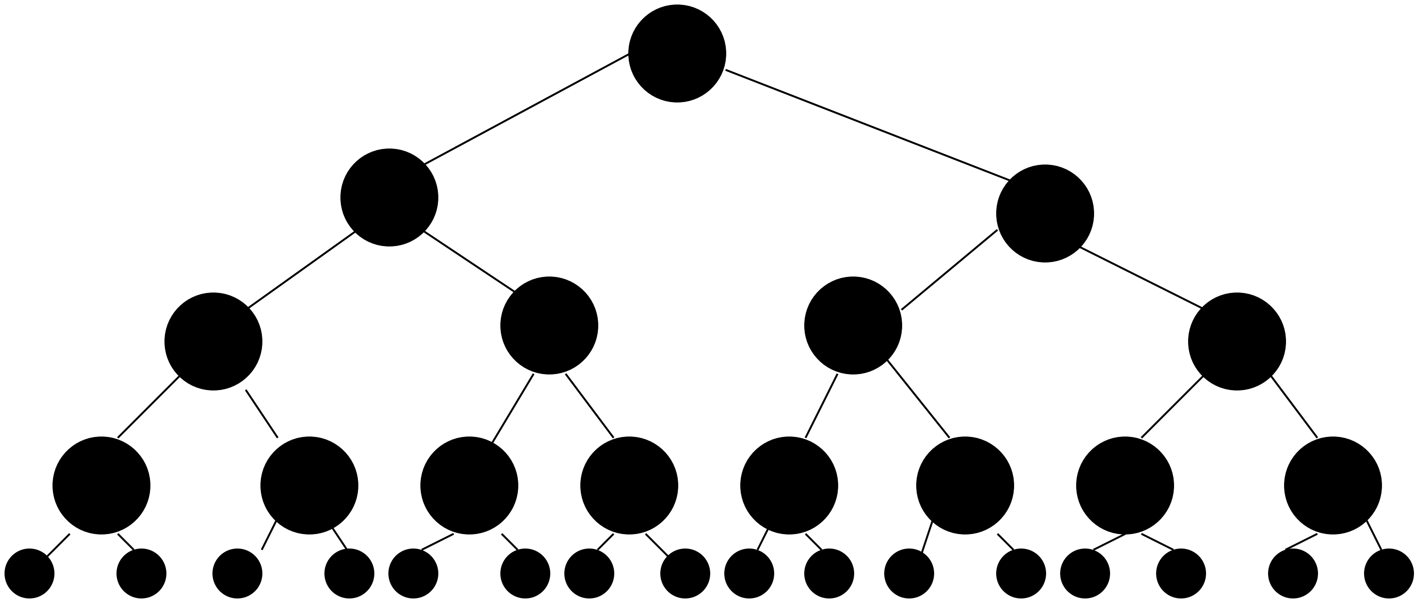
Red-Black Trees

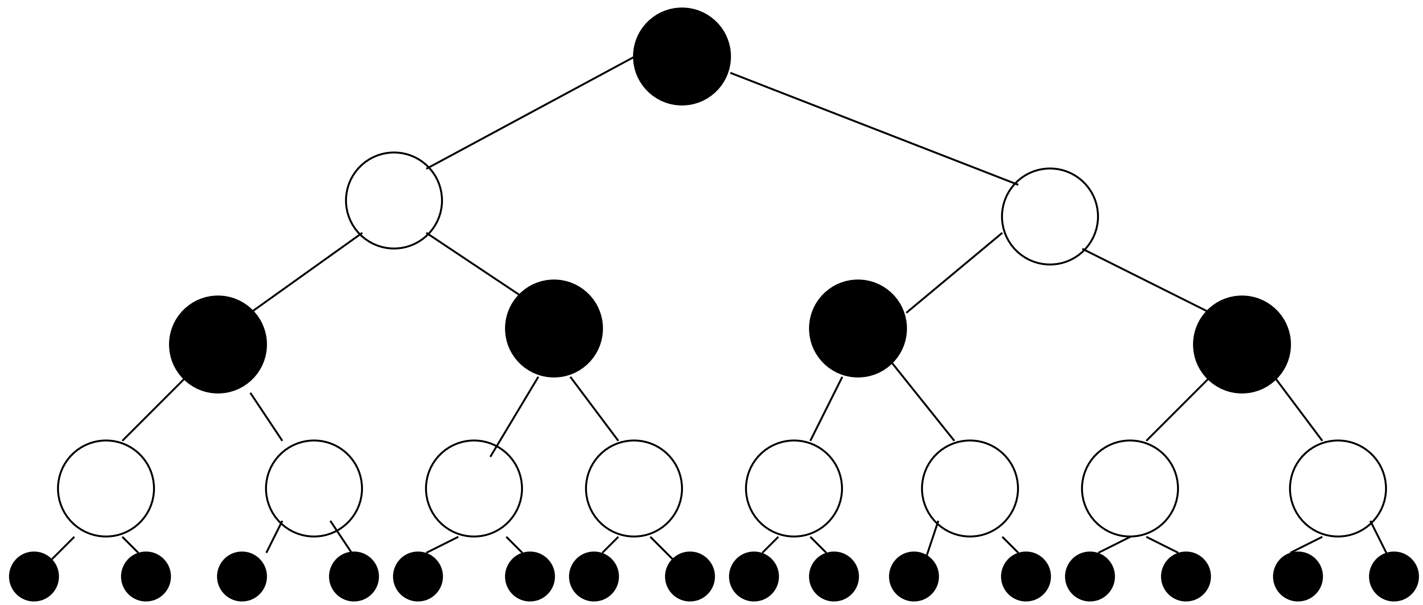
Red-Black Trees

- Definition: A red-black tree is a binary search tree where:
  - Every node is either red or black.
  - Each NULL pointer is considered to be a black "node"
  - If a node is red, then both of its children are black.
  - Every path from a node to a NULL contains the same number of black nodes.
  - The root is black
- Definition: The black-height of a node, X, in a red-black tree is the number of black nodes on any path to a NULL, not counting X.



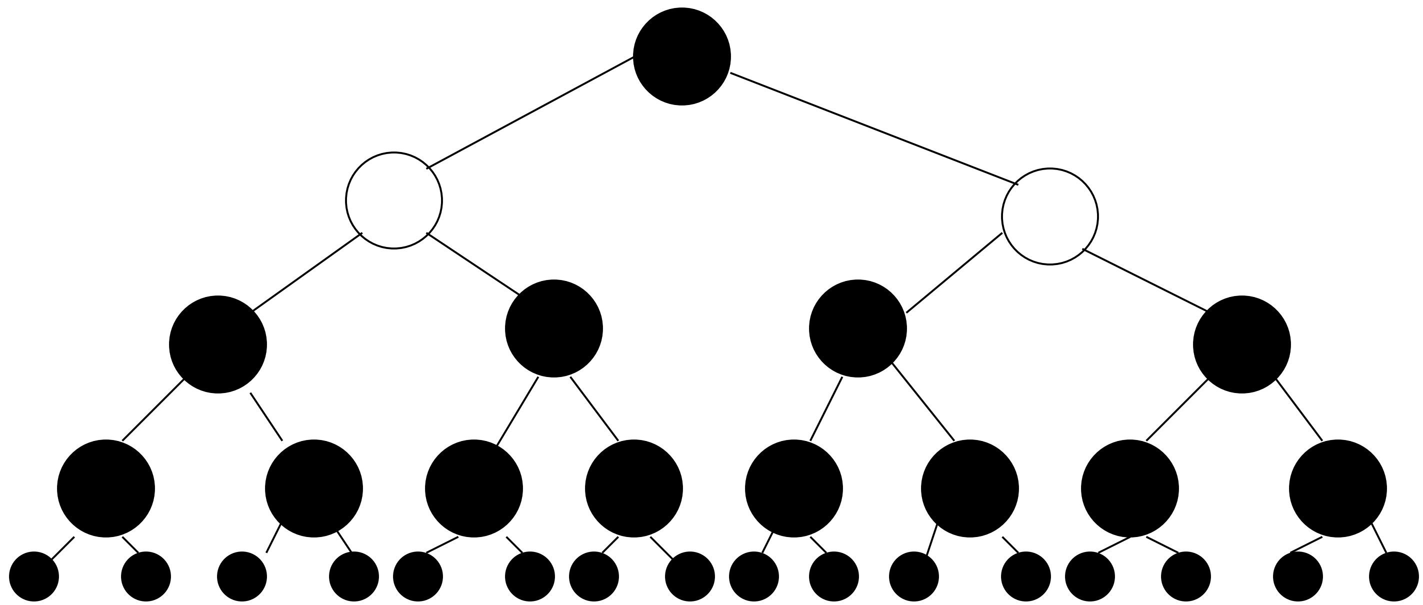
A Red-Black Tree with NULLs shown

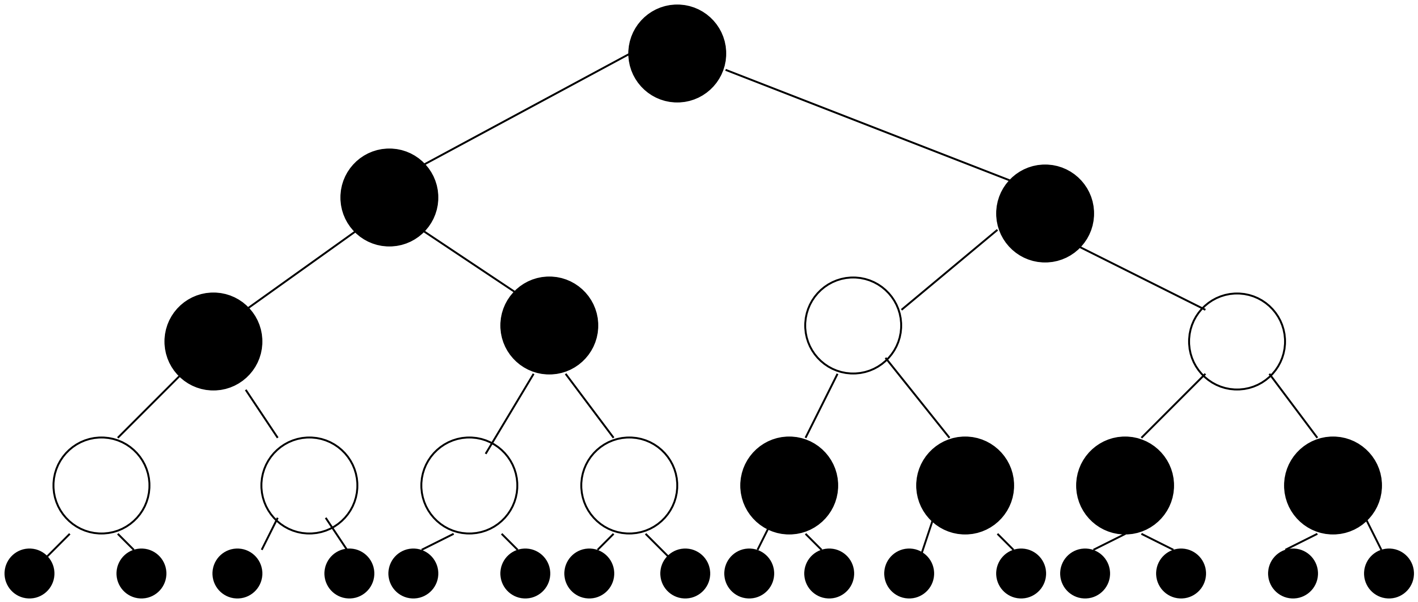
Black-Height of the tree = 4



A valid Red-Black Tree

Black-Height = 2





Theorem 1 – Any red-black tree with root ***x***, has **$n >= 2^{bh(x)} – 1$** nodes, where bh(x) is the black height of node x.

Proof: by induction on height of x.

Theorem 2 – In a red-black tree, at least half the nodes on any path from the root to a NULL must be black.

Proof – If there is a red node on the path, there must be a corresponding black node.

Algebraically this theorem means

$$bh(\ x\ ) \geq h/2$$

Theorem 3 – In a red-black tree, no path from any node, N, to a NULL is more than twice as long as any other path from N to any other NULL.

Proof: By definition, every path from a node to any NULL contains the same number of black nodes. By Theorem 2, a least _ the nodes on any such path are black. Therefore, there can no more than twice as many nodes on any path from N to a NULL as on any other path. Therefore the length of every path is no more than twice as long as any other path

Theorem 4 –

A red-black tree with ***n*** nodes has height

$$h \leq 2\lg(\boldsymbol{n} + 1).$$

Proof: Let h be the height of the red-black tree with root x. By Theorem 2,

$$bh(x) \geq h/2$$

From Theorem 1, $\boldsymbol{n} \geq 2^{bh(x)} - 1$

Therefore $n \geq 2^{h/2} - 1$

$$n + 1 \geq 2^{h/2}$$

$$\lg(n + 1) \geq h/2$$

$$2\lg(n + 1) \geq h$$

Bottom –Up Insertion

- Insert node as usual in BST
- Color the Node RED
- What Red-Black property <u>may</u> be violated?
  - Every node is Red or Black
  - NULLs are Black
  - If node is Red, both children must be Black
  - Every path from node to descendant NULL must contain the same number of Blacks

Bottom Up Insertion

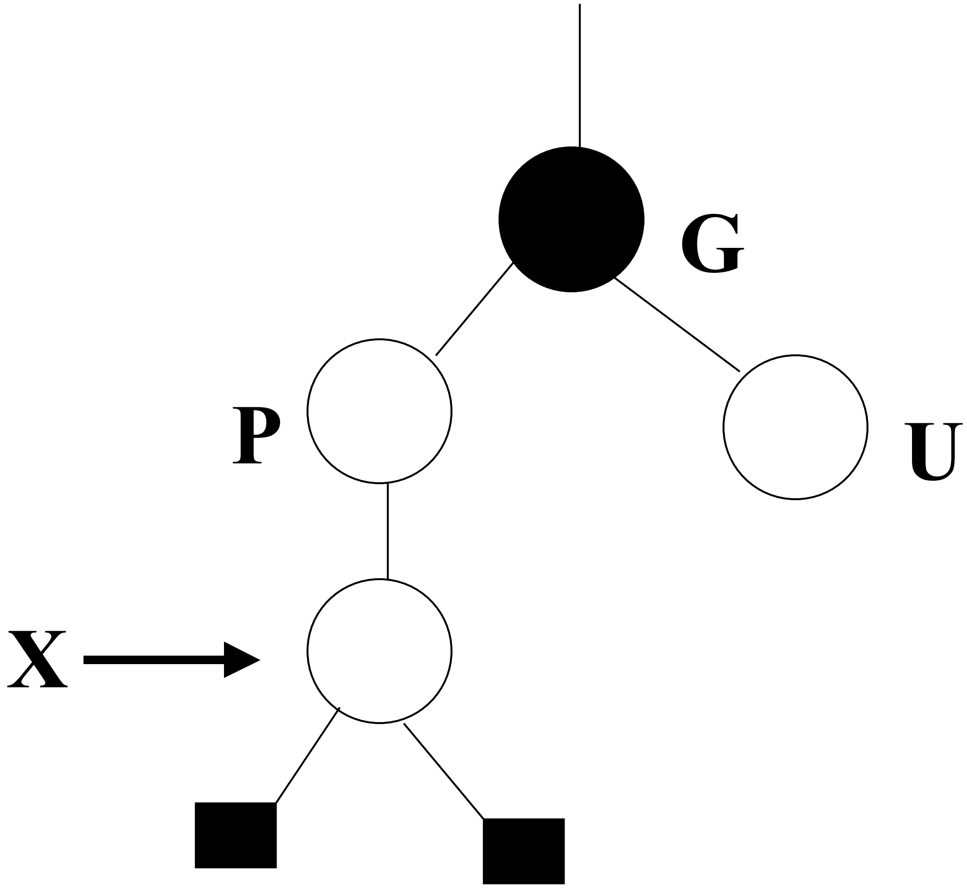
- Insert node; Color it RED; X is pointer to it
- Cases

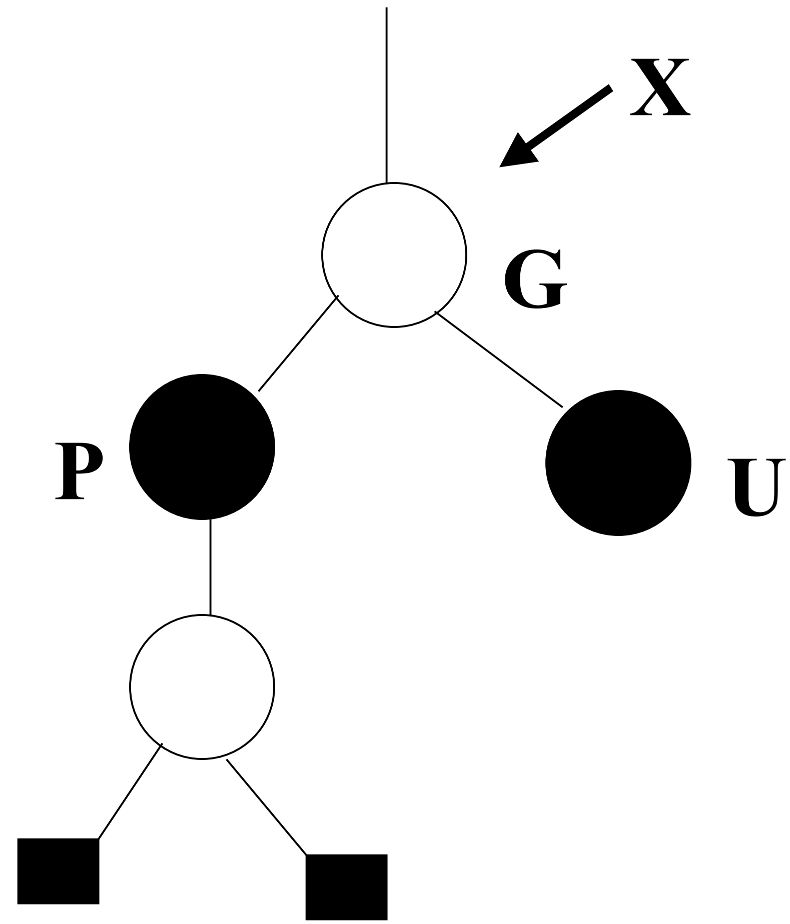
  0: X is the root -- color it black

  1: Both parent and uncle are red -- color parent and uncle black, color grandparent red, point X to grandparent, check new situation

  2 (zig-zag): Parent is red, but uncle is black. X and its parent are opposite type children -- color grandparent red, color X black, rotate left(right) on parent, rotate right(left) on grandparent

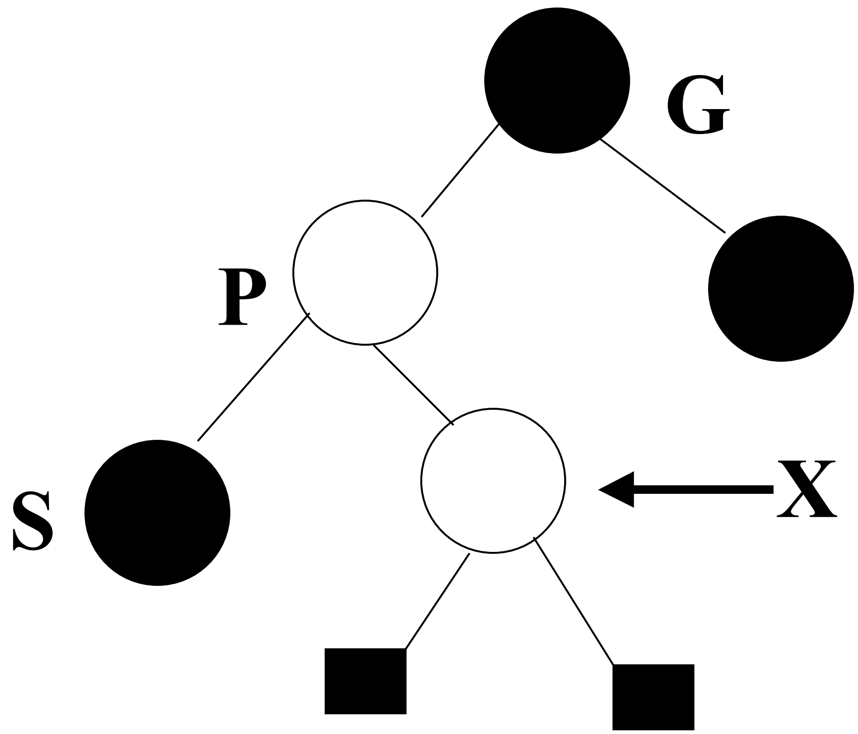
  3 (zig-zig): Parent is red, but uncle is black. X and its parent are both left (right) children -- color parent black, color grandparent red, rotate right(left) on grandparent

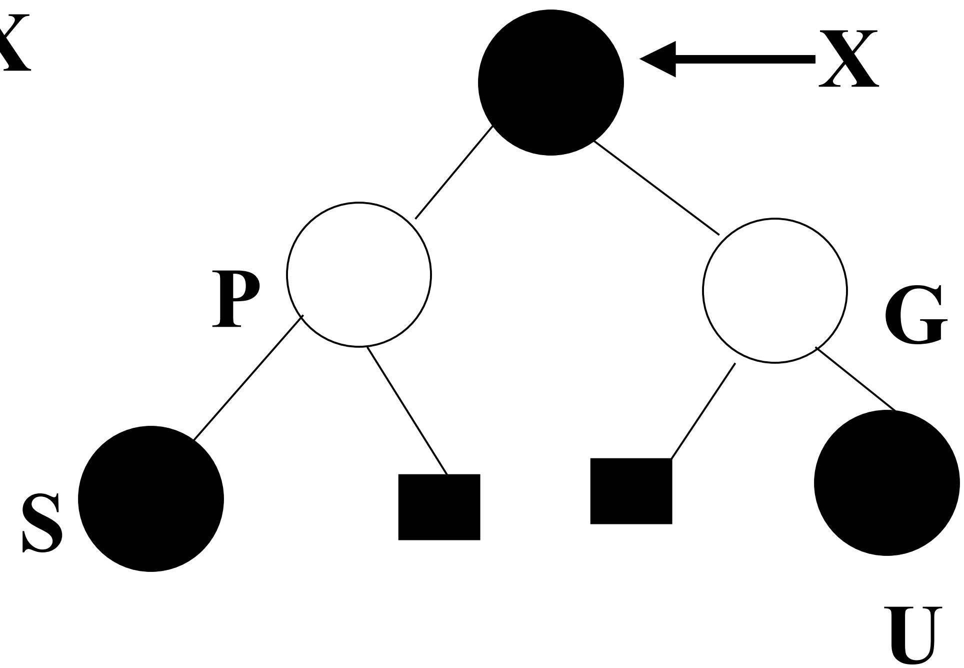




Case 1 – U is Red

Just Recolor and move up

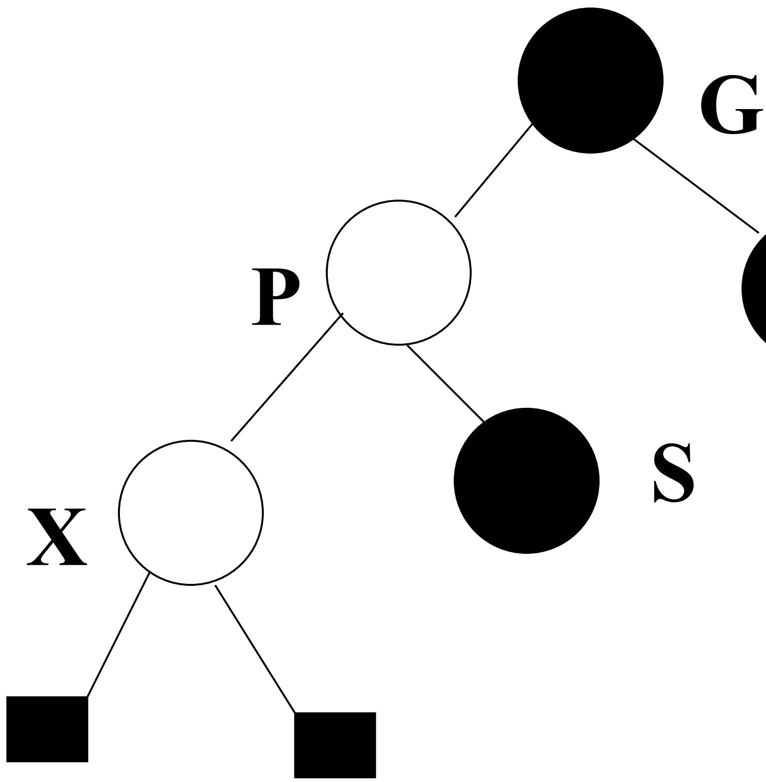




Case 2 – Zig-Zag

Double Rotate
  X around P; X around G

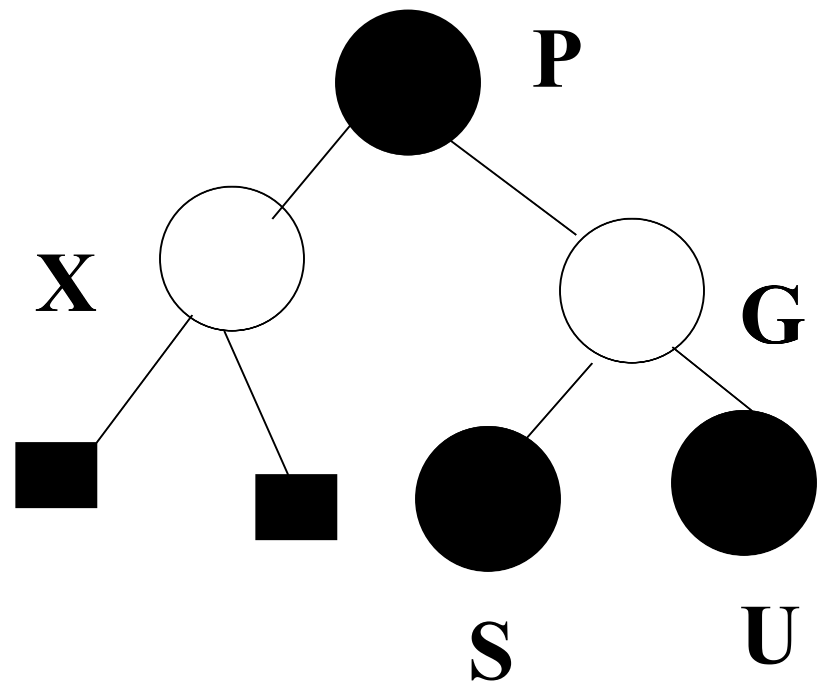
Recolor G and X



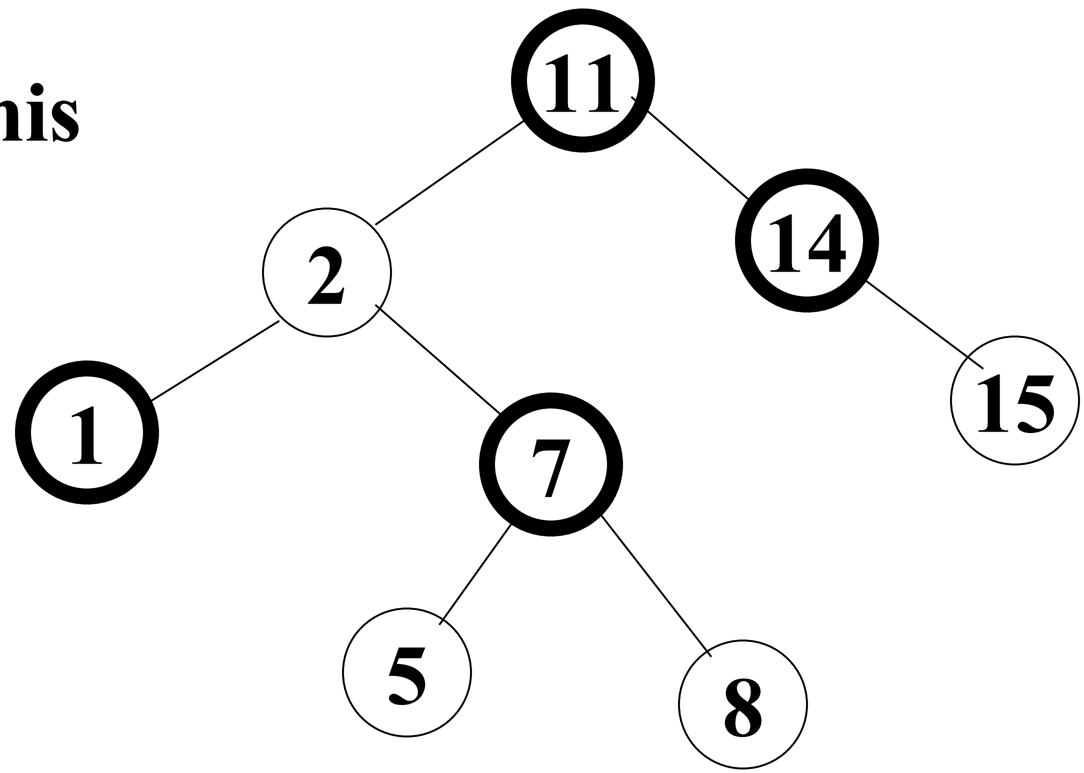
Case 3 – Zig-Zig

Single Rotate P around G

Recolor P and G



Insert 4 into this R-B Tree



Insertion Practice

Insert the values 2, 1, 4, 5, 9, 3, 6, 7 into an initially empty Red-Black Tree

Asymptotic Cost of Insertion

- O(lg n) to descend to insertion point
- O(1) to do insertion
- O(lg n) to ascend and readjust == worst case only for case 1

- Total: O(log n)